CLAIMS

1. A method for providing a reusable, run time configurable design test bench, the method comprising:

partitioning functionality of a test bench between a design verification engine and a scripting language;

implementing a library of one or more scripted routines that allow one or more interpreters to be instantiated in one or more verification engine test benches, wherein said library allows one or more interpreters to be instantiated in a verification engine simulation;

said one or more interpreters interacting with said simulation to cause tasks to be executed in said simulation, wherein said simulation starts up an interpreter and instructs it to run a script;

said interpreter passing control back to said verification engine so that said task can be executed when said interpreter encounters a function that is mapped to a certain verification task; and

resuming execution of said one or more scripted routines after executing said task.

2. The method of Claim 1, said method further comprising:

passing control between said verification engine and said one or more scripted routines through function calls on either side;

wherein said verification engine controls when said one or more interpreters are invoked; and

wherein said one or more interpreters cause verification engine tasks to be executed while retaining a state of said one or more interpreters, wherein said one or more interpreters resume execution of said script after said verification engine task is completed.

3. The method of Claim 1, wherein said one ore more interpreters comprise a Tcl server that runs on a separate thread from said simulation.

4. The method of Claim 1, wherein said verification engine is a Verilog engine.

5. The method of Claim 1, said method further comprising:

synchronizing said simulation and said one or more interpreters via one or more semaphores;

wherein control is passed freely between said verification engine and said one or more interpreters.

6. The method of Claim 1, said method further comprising:

providing a verification engine module for determining when said one or more interpreters are invoked; and

causing verification engine tasks to be executed with said one or more interpreters.

7. The method of Claim 6, wherein a function call executes a script, a coded function call executes a verification engine task, and a function call resumes script execution.

8. The method of Claim 1, wherein one or more verification engine tasks have access to arguments passed to said one or more interpreter functions that invoked said task, wherein information is passed from said one or more interpreters to said verification engine.

9. The method of Claim 1, wherein said one or more verification engine tasks control return values of their one or more interpreter counterparts, wherein information is passed from said verification engine to said one or more interpreters.

10. The method of Claim 1, said method further comprising:

providing one or more routines for direct sharing of information between any of said one or more interpreters and said verification engine, and between different interpreters.

11. The method of Claim 1, wherein said library comprises a TCL_PLI library.

12. The method of Claim 11, wherein said TCL_PLI library comprises any of the following PLI functions: $tclInit, $tclExec, $tclGetArgs, and $tclClose.

13. The method of Claim 12, wherein said $tclInit function creates and initializes a new Tcl interpreter, defines new Tcl functions that are used to invoke Verilog tasks, maps said functions to specific tasks, and defines how many arguments said functions take.

14. The method of Claim 12, wherein said $tclExec function passes control from Verilog to a Tcl server, launches a new script, or resumes execution of a script that was stalled when an interpreter encountered a function that was mapped to a Verilog task.

15. The method of Claim 14, wherein said $tclExec function returns under one of the following conditions: when an error occurs, when a script ends, or when a function is encountered that is mapped to a Verilog task.

16. The method of Claim 12, wherein said $tclGetArgs function accesses argument values that were passed to an extended Tcl function.

17. The method of Claim 12, wherein said $tclClose function destroys a Tcl interpreter and frees resources with which it is associated.

18. The method of Claim 11, said TCL_PLI library further comprising:

a $tclLinkVariables function for allowing direct sharing of variables between Verilog and Tcl and linking a list of Verilog variables with a list of Tcl variables;

wherein said TCL_PLI library then automatically keeps these variables synchronized until an interpreter is deleted;

wherein said $tclLinkVariables function supports integer and string variables, and can mark variables as read-only in said Tcl interpreter;

wherein said $tclShareVariables function allows direct sharing of variables between two different Tcl interpreters, without any connection to Verilog; and

wherein, after a call to said $tclShareVariables function, a list of Tcl variables in both interpreters is automatically synchronized by a TCL_PLI library, until one of said interpreters is deleted.

19. The method of Claim 11, said TCL_PLI library further comprising:

a $tclSetMCD function; and

a $tclAddMCD function;

wherein said $tclSetMCD function, and a $tclAddMCD function allow a Tcl interpreter access to multi-channel descriptors in a Verilog simulation and, thereby, allow a user to redirect messages from a Tcl interpreter into log files that also record messages directly from said simulation, wherein the order in which messages were generated is preserved.

20. The method of Claim 11, said TCL_PLI library further comprising:

a $tclSetErrorReg function for allowing a user to identify one register in a Verilog simulation that is linked to any error occurring in any interpreter or in TCL_PLI;

wherein, if any error occurs, a value of said register is changed, thereby allowing said simulation to react to said error immediately.

21. The method of Claim 11, said TCL_PLI library further comprising:

a $tclWarnOnX function for causing TCL_PLI to print a warning message under a predefined conditions;

wherein execution of said one or more scripts continues.

22. The method of Claim 1, said method further comprising:

providing a module that instantiates Synopsys LMC source models for a PCI master and a PCI slave together with at least one Tcl interpreter.

23. The method of Claim 21, wherein tasks supplied with PCI models are mapped to extended Tcl functions, allowing execution of Tcl scripts that interact with other devices on a PCI bus

24. The method of Claim 21, wherein Verilog code causes one of said Tcl interpreters to start executing a script when it senses an interrupt on a PCI bus.

25. The method of Claim 21, said method further comprising:

checking whether a PCI bus is busy before calling a task that starts a transaction on said bus; and

if said bus is busy, waiting for the current transaction to complete before starting a new transaction.

26. The method of Claim 21, said method further comprising:

providing a PCI_TCL module for extensive testing of any PCI based device without writing a single line of Verilog code for said test bench.

27. The method of Claim 21, said method further comprising:

providing a library of Tcl procedures that simplifies tasks.

28. The method of Claim 21, wherein functionality between Tcl and Verilog is partitioned such that said scripts are reusable in system level testing.

29. The method of Claim 21, said method further comprising:

providing a porting of Tcl scripts to real hardware; and

running a verification suite on ASICs when they return from a foundry.

30. An apparatus for providing a reusable, run time configurable design test bench, comprising:

a design verification engine; and

a scripting language;

wherein functionality of a test bench is partitioned between said design verification engine and said scripting language.

31. The apparatus of Claim 29, further comprising:

a verification engine simulation;

one or more interpreters;

a library of one or more scripted routines that allow said one or more interpreters to be instantiated in one or more verification engine test benches;

wherein said library allows one or more interpreters to be instantiated in said verification engine simulation;

said one or more interpreters interacting with said simulation to cause tasks to be executed in said simulation, wherein said simulation starts up an interpreter and instructs it to run a script;

wherein said interpreter passes control back to said verification engine so that said task can be executed when said interpreter encounters a function that is mapped to a certain verification task; and

a mechanism for resuming execution of said one or more scripted routines after executing said task.

32. The apparatus of Claim 29, further comprising:

a module for passing control between said verification engine and said one or more scripted routines through function calls on either side;

wherein said verification engine controls when said one or more interpreters are invoked; and

wherein said one or more interpreters cause verification engine tasks to be executed while retaining a state of said one or more interpreters, wherein said one or more interpreters resume execution of said script after said verification engine task is completed.

33. The apparatus of Claim 29, wherein said one ore more interpreters comprise a Tcl server that runs on a separate thread from said simulation.

34. The apparatus of Claim 29, wherein said verification engine is a Verilog engine.

35. The apparatus of Claim 29, further comprising:

one or more semaphores for synchronizing said simulation and said one or more interpreters;

wherein control is passed freely between said verification engine and said one or more interpreters.

36. The apparatus of Claim 29, further comprising:

a verification engine module for determining when said one or more interpreters are invoked; and

a module for causing verification engine tasks to be executed with said one or more interpreters.

37. The apparatus of Claim 35, further comprising:

a first function call for executing a script;

a coded function call for executing a verification engine task; and

a second function call for resuming script execution.

38. The apparatus of Claim 29, further comprising:

one or more routines for direct sharing of information between any of said one or more interpreters and said verification engine, and between different interpreters.

39. The apparatus of Claim 29, wherein said library comprises a TCL_PLI library.

40. The apparatus of Claim 38, wherein said TCL_PLI library comprises any of the following PLI functions: $tclInit, $tclExec, $tclGetArgs, and $tclClose.

41. The apparatus of Claim 39, wherein said $tclInit function creates and initializes a new Tcl interpreter, defines new Tcl functions that are used to invoke Verilog tasks, maps said functions to specific tasks, and defines how many arguments said functions take.

42. The apparatus of Claim 39, wherein said $tclExec function passes control from Verilog to a Tcl server, launches a new script, or resumes execution of a script that was stalled when an interpreter encountered a function that was mapped to a Verilog task.

43. The apparatus of Claim 41, wherein said $tclExec function returns under one of the following conditions: when an error occurs, when a script ends, or when a function is encountered that is mapped to a Verilog task.

44. The apparatus of Claim 39, wherein said $tclGetArgs function accesses argument values that were passed to an extended Tcl function.

45. The apparatus of Claim 39, wherein said $tclClose function destroys a Tcl interpreter and frees resources with which it is associated.

46. The apparatus of Claim 38, said TCL_PLI library further comprising:

a $tclLinkVariables function for allowing direct sharing of variables between Verilog and Tcl and linking a list of Verilog variables with a list of Tcl variables;

wherein said TCL_PLI library then automatically keeps these variables synchronized until an interpreter is deleted;

wherein said $tclLinkVariables function supports integer and string variables, and can mark variables as read-only in said Tcl interpreter;

wherein said $tclShareVariables function allows direct sharing of variables between two different Tcl interpreters, without any connection to Verilog; and

wherein, after a call to said $tclShareVariables function, a list of Tcl variables in both interpreters is automatically synchronized by a TCL_PLI library, until one of said interpreters is deleted.

47. The apparatus of Claim 38, said TCL_PLI library further comprising:

a $tclSetMCD function; and

a $tclAddMCD function;

wherein said $tclSetMCD function, and a $tclAddMCD function allow a Tcl interpreter access to multi-channel descriptors in a Verilog simulation and, thereby, allow a user to redirect messages from a Tcl interpreter into log files that also record messages directly from said simulation, wherein the order in which messages were generated is preserved.

48. The apparatus of Claim 38, said TCL_PLI library further comprising:

a $tclSetErrorReg function for allowing a user to identify one register in a Verilog simulation that is linked to any error occurring in any interpreter or in TCL_PLI;

wherein, if any error occurs, a value of said register is changed, thereby allowing said simulation to react to said error immediately.

49. The apparatus of Claim 38, said TCL_PLI library further comprising:

a $tclWarnOnX function for causing TCL_PLI to print a warning message under a predefined conditions;

wherein execution of said one or more scripts continues.

50. The apparatus of Claim 29, further comprising:

a module that instantiates Synopsys LMC source models for a PCI master and a PCI slave together with at least one Tcl interpreter.

51. The apparatus of Claim 49, further comprising:

a PCI_TCL module for extensive testing of any PCI based device without writing a single line of Verilog code for said test bench.

52. The apparatus of Claim 49, further comprising:

a library of Tcl procedures that simplifies tasks.

53. The apparatus of Claim 49, further comprising:

a porting of Tcl scripts to real hardware; and

a verification suite for running on ASICs when they return from a foundry.